VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2019 František Šumšala

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student:	**František Šumšala**
Studijní program:	B2647 Informační a komunikační technologie
Studijní obor:	2612R025 Informatika a výpočetní technika
Téma:	Absolvování individuální odborné praxe Individual Professional Practice in the Company
Jazyk vypracování:	čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Craneballs s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Martin Němec, Ph.D.**

Konzultant bakalářské práce: Ing. Matěj Rejnoch

Datum zadání: 01.09.2018
Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2019 ..... Tumiala .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 18. dubna 2019 .....

Na tomto místě bych rád poděkoval firmě Craneballs s.r.o. za umožnění mi absolvovat individuální odbornou praxi. Také bych chtěl poděkovat jejímu kolektivu a hlavně Ing. Lukáši Krislovi a Ing. Tomáši Popkovi za mé vedení při jejím vykonávání.

Abstrakt

Tato práce se zabývá popisem autorem absolvované individuální odborné praxe u firmy Craneballs s.r.o., která se zabývá vývojem videoher. Během této praxe byl autor zařazen do čtyřčlenného studentského týmu, s nímž pracoval na hře pro virtuální realitu, specificky platformu SteamVR, za využití herního enginu Unity. Práce čtenáře blíže seznamuje s problematikou vývoje her pro tuto platformu a popisuje úkoly, na jejichž řešení autor v rámci tohoto projektu pracoval.

Klíčová slova: Craneballs, vývoj her, Unity, virtuální realita, VR, SteamVR, Virtual Reality Toolkit, VRTK

Abstract

This thesis details author's internship at game development studio Craneballs s.r.o. During this internship author worked as part of a four man student team on a game for virtual reality, specifically SteamVR platform, utilizing the Unity game engine. This paper acquaints the reader with the issues faced when developing games for this platform and describes the tasks assigned to the author while working on this project.

Key Words: Craneballs, game development, Unity, virtual reality, VR, SteamVR, Virtual Reality Toolkit, VRTK

Obsah

Seznam použitých zkratek a symbolů

2D	–	dvourozměrný
3D	–	trojrozměrný
FPS	–	First Person Shooter
PC	–	osobní počítač
PvP	–	Player versus Player
UI	–	uživatelské rozhraní
UML	–	Unified Modeling Language
VR	–	virtuální realita
VRTK	–	Virtual Reality Toolkit

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Cílem této bakalářské práce je popis mého působení ve firmě Craneballs s.r.o. [1] (dále jen Craneballs) v rámci absolvování individuální odborné praxe. Společnost Craneballs se sídlem v Ostravě se primárně zabývá vývojem mobilních her, dále i vývojem her počítačových a her pro virtuální realitu. Možnost absolvování individuální odborné praxe v této firmě jsem si zvolil pro svůj osobní zájem o odvětví vývoje her a pro získání cenných praktických zkušeností.

Během této praxe jsem byl zařazen do čtyřčlenného studentského týmu, jehož cílem bylo podle předloženého zadání vytvořit hru pro virtuální realitu, specificky platformu SteamVR [2]. Jedná se o hru pro jednoho hráče, kde jeho cílem je se co nejdéle ubránit vlnám útočících nepřátel a dosáhnout tak co nejvyššího skóre. Dané skóre je poté zapsáno do online tabulky výsledků a lze jej tak porovnat s ostatními hráči. Tato hra byla vytvořena za využití herního enginu Unity [3] a pro něj určené knihovny Virtual Reality Toolkit [4]. V tomto projektu jsem vystupoval jako gameplay programátor a mé úkoly byly primárně zaměřeny na zprovoznění virtuální reality a implementaci interaktivních prvků hry a systémů určených pro její řízení.

V této práci je obecně popsána problematika vývoje her se specifickým zaměřením na hry pro virtuální realitu. Jedná se například o problematiku pohybu ve virtuální realitě nebo možnosti interakce s jinými objekty uvnitř tohoto prostředí. Je blíže popsán herní engine Unity a proces vývoje her v něm. Taktéž je přiblížena knihovna Virtual Reality Toolkit a její praktické využití.

Na začátku práce je čtenář blíže seznámen s firmou Craneballs, mým zařazením do projektu a vyvíjenou hrou samotnou. Posléze jsou popsány konkrétní úkoly, na jejichž řešení jsem během vývoje pracoval. U jednotlivých úkolů je nastíněna jejich daná problematika a mnou implementované řešení. V závěru práce jsou shrnuty znalosti nabyté během studia, které jsem při praxi využil a také znalosti, které mi scházely. Jsou zhodnoceny dosažené výsledky a praxe samotná.

2 Popis odborného zaměření firmy a pracovního zařazení

Firma Craneballs byla založena v roce 2009 jako nezávislé herní studio sídlící v Ostravě. Jejím původním zaměřením byl vývoj mobilních her pro platformu iOS. V prvních dvou letech studio vydalo několik titulů, avšak nejmarkantnějšího úspěchu dosáhlo v roce 2011 se svou hrou Overkill [5]. Po tomto úspěchu se studio pevně usadilo na herním trhu a začalo se věnovat i vývoji her pro platformu Android. Dodnes studio vydalo téměř dvacítku mobilních her, mezi které patří například hry Bomb Hunters [6][7], Splash Cars [8][9] a nebo už druhé pokračování série Overkill s názvem Overkill 3 [10][11].

V současnosti studio pracuje na mobilní PvP hře Medieval Smackdown [12][13] a ve vývoji je první počítačová hra studia s názvem Planet Nomads [14]. Jedná se o sci-fi sandbox survival hru z pohledu první osoby, jejíž vývoj byl podpořen úspěšnou Kickstarter kampaní v roce 2016.[15] V neposlední řadě se studio v posledních letech začalo zabývat i vývojem her pro virtuální realitu, specificky platformu SteamVR.

Dnes už studio pro svůj veškerý vývoj využívá herního enginu Unity, poskytující silné nástroje pro tvorbu 2D a 3D her pro všechny komerčně úspěšné platformy.

2.1 Popis pracovního zařazení

V rámci vyvíjené hry jsem byl zařazen do čtyřčlenného studentského týmu, skládajícího se ze dvou programátorů a dvou grafiků. Tento tým pracoval pod vedením tří zaměstnanců z firmy Craneballs. Tým jako celek byl veden přiděleným game designérem, který vytvořil původní vizi hry, v průběhu vývoje určoval její případně změny a zadával jednotlivým členům týmu úkoly. S druhým programátorem v týmu jsme byli dále pod vedením jednoho z programátorů, který nás zaučil v programování her v enginu Unity a řešil s námi případné problémy při řešení úkolů. Grafikům byl pro stejné účely přidělen vedoucí grafik.

Během vývoje jsem v týmu zastupoval roli hlavního gameplay programátora. Mé úkoly byly soustředěny na zprovoznění virtuální reality a komunikaci s rozhraním daného zařízení skrze knihovnu Virtual Reality Toolkit. Dále na implementování mechanik umožňující hráči interagovat s ostatními objekty ve virtuálním prostředí a v neposlední řadě na vytvoření systémů, které hru na pozadí řídí.

3 Zadání projektu

Našemu týmu bylo dáno úkolem vytvořit hru pro jednoho hráče, určenou pro virtuální realitu. Tým od vedoucího designéra dostal k dispozici takzvaný game design document, popisující o jakou hru se má jednat, co má obsahovat, čím je inspirována a jak by měl v jednotlivých iteracích probíhat její vývoj. Tým měl také možnost vyzkoušet demo hry se základní funkcionalitou, kterou dříve zaměstnanci firmy vytvořili. To firmě sloužilo pro otestování původní vize hry a týmu jako předloha pro její počáteční vývoj.

Hra měla být vyvinuta za využití herního enginu Unity, jelikož se jedná o engine, ve kterém firma už několik let vyvíjí veškeré své hry. Dále měla být zužitkována knihovna Virtual Reality Toolkit, tak aby nebylo nutné vytvářet vlastní implementaci komunikace se zařízením virtuální reality.

3.1 Popis game designu hry

Týmem vyvíjená hra, později pojmenována jako 'Drone Wars VR', je hra pro jednoho hráče, založená na herním žánru FPS. Hráč je umístěn v jedné z arén a jeho cílem je se po co nejdelší dobu ubránit útokům nepřátel. K tomu mu slouží dva disky, kterými může blokovat nepřátelské útoky a také jimi může po nepřátelích vrhat s cílem jejich zničení. Při dosažení nejlepšího osobního skóre pro danou arénu a herní mód je toto skóre uloženo do online tabulky výsledků. V této tabulce lze poté porovnat své výsledky s ostatními hráči a snažit se jejich skóre překonat.

Hra obsahuje několik různých arén a herních módů. V jednotlivých kombinacích arény a módu se mohou vyskytovat různé typy nepřátel nebo jejich různé varianty s odlišným nastavením. Hru by mělo být možné dále rozšířit přidáním nových prvků, jako příklad lze uvést třeba interaktivní pickupy ze zničených nepřátel, při jejichž zasažení je hráči obnovena část životů nebo dán krátkodobý zesilující efekt.

Po vizuální stránce je hra zařazena do žánru sci-fi fantasy a je silně inspirována filmem Tron: Legacy [16]. Jiné herní tituly, kterými je hra taktéž inspirována jsou hry Sparc [17], Space Pirate Trainer [18] a Blue Effect [19].

3.2 Herní módy

Všechny módy hry pracují na stejném základu, kdy hráč má omezený počet životů, při jejichž vypršení hra končí. Základním módem je mód *Endless*, ve kterém skóre hráče závisí na počtu zničených nepřátel.

Tento základ je rozšířen módy, které do hry přidávají čas. Mód *High Ground* je podobný módu *Endless*, avšak zde má hráč omezený čas, ve kterém musí zničit co nejvíce nepřátel. V módu *Time Attack* hráč začíná hru s velice omezeným časem a za každého zničeného nepřítele dostává čas navíc. Zde je výsledné skóre dáno časem po který se hráč udržel ve hře.

4 Seznam úkolů zadaných v průběhu odborné praxe

Následující seznam popisuje hlavní úkoly, na jejichž řešení jsem během praxe pracoval. Popsané úkoly jsou seřazeny s ohledem na jejich časovou posloupnost tak, aby byl co nejpřesněji reprezentován vývoj samotné hry:

1. **Seznámení se s knihovnou VRTK a jejími základními prvky** - Získání zkušeností s možnými interaktivními prvky VR her a identifikace prvků vhodných pro vyvíjenou hru. Pochopení komponent, kterými jsou tvořeny a ovládány pomocí knihovny Virtual Reality Toolkit. Následné přenesení těchto prvků do základní scény pro ověření jejich správného fungování.

2. **Implementace základní funkcionality disků** - Realizace pohybu disků, jejich rotace, odrazů od levelu arény a jejich přivolání zpět k hráči.

3. **Projektily, atributy hráče a nepřátel a jejich úmrtí** - Vytvoření projektilů, které mohou nepřátelé střílet po hráči a jejich odrážení hráčem. Implementace systému pro správu atributů hráče a nepřátel. Reakce na zásah projektilem jejich zraněním nebo úmrtím.

4. **Opravy nechtěných interakcí a přepracování načítaní levelů** - Využití párového programování k odstranění nežádoucích efektů ovlivňující hráče a úpravě systémů pro načtení levelu pro jejich lepší funkcionalitu v budoucnosti.

5. **Přidání doměřování disků** - Realizace doměřování disků jakožto mechanizmu snižující nepřesnost hodů.

6. **Refaktorizace disků a s ní související přístup k architektuře** - Adaptování komponentní a událostmi řízené architektury pro zvýšení kvality kódu. Jejich aplikování na disky.

7. **Nahrazení disků štíty a vrhanými koulemi** - Snaha o vyřešení nepřesnosti disků při různých typech hodů. Implementace štítu a vrhané koule pro jejich nahrazení.

8. **Finální ladění a balancování hry** - Testování hry pro vhodné nastavení jednotlivých prvků za účelem zlepšení herního zážitku. Úpravy hry na základě zpětné vazby.

Tabulka 1: Přibližná časová náročnost zadaných úkolů

Úkol	Čas (ve dnech)
Seznámení se s knihovnou VRTK a jejími základními prvky	2
Implementace základní funkcionality disků	4
Projektily, atributy hráče a nepřátel a jejich úmrtí	5
Opravy nechtěných interakcí a přepracování načítaní levelů	4
Přidání doměřování disků	6
Refaktorizace disků a s ní související přístup k architektuře	7
Nahrazení disků štíty a vrhanými koulemi	12
Finální ladění a balancování hry	6

5 Využité technologie a metodiky

5.1 Unity

V minulosti taktéž nazývaný jako Unity3D, je herní engine společnosti Unity Technologies určený pro komerční vývoj 2D a 3D her. Poskytuje řadu nástrojů pro vývoj her nejrůznějších žánrů a cílových platforem, zahrnující PC, web, konzole, mobilní zařízení a virtuální realitu. [20] Engine je volně dostupný zdarma, umožňující bezbariérový přístup pro nadšence, studenty a začínající vývojáře. Pro už úspěšné vývojáře a společnosti jsou k dispozici placené verze obsahující další nástroje, nazvané jako Unity Plus a Unity Pro.[21]

První verze Unity byla vydána v roce 2005 s omezeným počtem podporovaných platforem a byla dostupná pouze pro Mac OS X.[22] S dalším vývojem byl rozšířen počet platforem pro které bylo možné hry vytvářet a byla vytvořena verze enginu pro operační systém Windows. Engine v minulosti podporoval programování her v jazyce Boo a speciální verzi jazyka JavaScript nazvanou UnityScript, jejich podpora byla však ukončena v roce 2017.[23] Od tohoto roku je tedy možné programování her v Unity pouze za využití jazyka C#.

Pro vývoj hry popsané v této práci byla využita verze Unity 2018.2.5, jakožto v době zahájení jejího vývoje nejaktuálnější stabilní verze. Jelikož verze vydané během vývoje hry neobsahovaly novou funkcionalitu kritickou pro její vývoj a z důvodu potencionálního vzniku komplikací způsobených aktualizací Unity, nebyla využitá verze změněna.

5.2 Virtual Reality Toolkit

Virtual Reality Toolkit, zkráceně jako VRTK, je knihovna určená pro vývoj her pro virtuální realitu v enginu Unity. Jejím cílem je poskytnout abstraktní vrstvu nad nejrůznějšími VR technologiemi, která řeší samotné zpracování dat z daného zařízení. Tímto umožňuje snazší a rychlejší vývoj aplikací pro tuto platformu.

Knihovna VRTK byla vytvořena a je spravována uživatelem *TheStoneFox* a je volně dostupná skrze jeho GitHub repozitář [24] nebo Unity Asset Store [4].

5.3 Git

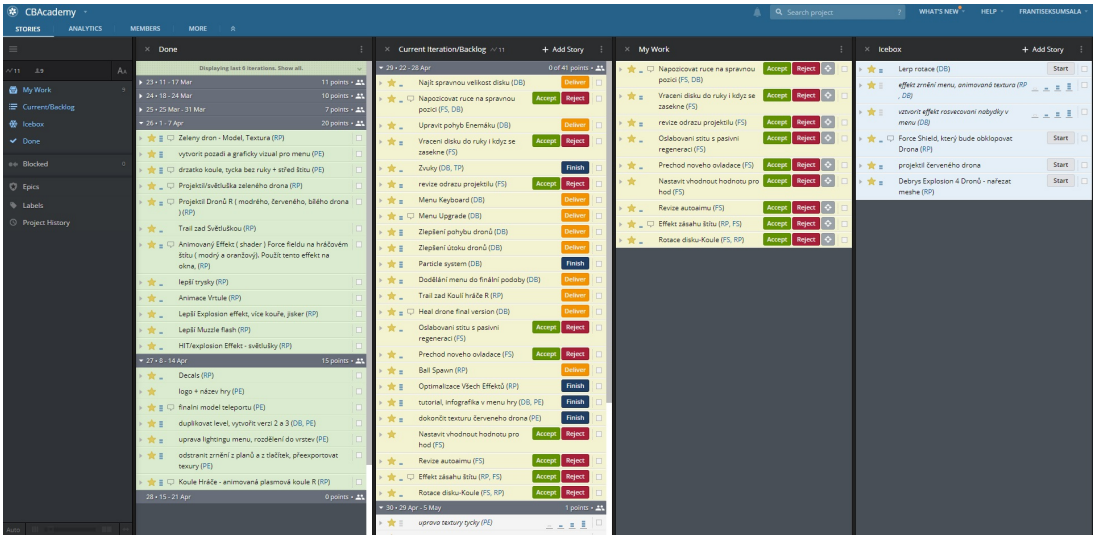
Git je dnes už široce rozšířený verzovací systém určený pro koordinaci týmového vývoje softwaru, jenž umožňuje sledování jednotlivých změn a navrácení k předešlým verzím.

Ačkoli jsem tuto technologii před začátkem praxe znal, základy jejího praktického použití jsem se naučil až během jejího absolvování. Pro účely vývoje hry byl vytvořen repozitář na GitHub [25] a tým během vývoje používal Git klienta Sourcetree [26].

5.4 Scrum

Pro řízení projektů se ve studiu Craneballs využívá agilní metody Scrum, která klade důraz na schopnost reagovat na změny v požadavcích. V rámci této metody se pracuje na základě iterací, kdy na počátku každé iterace jsou mezi jednotlivé členy týmu rozděleny úkoly a po určité době dojde k jejich vyhodnocení. V případě nevyhotovení úkolu je tento úkol přesunut do následující iterace.[27][28][29]

Tento přístup nám umožnil na základě určité vize vytvořit danou část hry, otestovat ji a určit, zdali je daná implementace vhodná a případně ji nahradit. Jednotlivé iterace většinou trvaly dva až tři týdny, kdy po ukončení dané iterace následovala prezentace nové verze hry zaměstnancům firmy. Pro účely zadávání a odevzdávání úkolů byl využit systém pro řízení projektů, který firma používá pro své projekty, s názvem Pivotal Tracker [30].



Obrázek 1: Řízení projektu na Pivotal Tracker

5.5 Párové programování

Párové programování, jak už název naznačuje, je metodika, při které dva programátoři společně pracují na dané části vyvíjeného softwaru. Obvykle se jedná o spolupráci trvající několik hodin s cílem zvýšení kvality výsledného řešení a prevence vzniku potencionálních chyb.

Při párovém programování oba programátoři pracují u jednoho zařízení a mají mezi sebou rozděleny role takzvaného řidiče a navigátora. Řidič se soustředí na psaní samotného kódu, jehož správnost kontroluje navigátor. Vzájemnou komunikací je nalezeno nejvhodnější řešení a jsou eliminovány chyby, kterých by se programátoři individuálně dopustili.[31][32]

Společně s druhým programátorem jsme tuto metodiku využili vícekrát. Primárně se jednalo o případy, kdy bylo potřeba propojit naše individuálně vytvořené části funkcionality hry nebo opravit velmi závažnou chybu.

6 Zvolený postup řešení zadaných úkolů

6.1 Seznámení se s knihovnou VRTK a jejími základními prvky

Po instalaci veškerého potřebného softwaru a seznámení se s game designem vyvíjené hry bylo mým prvním úkolem pochopit základy využití knihovny VRTK. Assety této knihovny obsahují desítky ukázkových scén, ve kterých jsou představeny nejrůznější ovladatelné prvky a možnosti interakce s herními objekty. Nejprve jsem tedy prošel některé z těchto scén, kde jsem osobně vyzkoušel chování těchto prvků ve virtuální realitě a poté v dané scéně zkoumal, z jakých komponent se skládají a čím jsou dány rozdíly jejich případných variant.

Jelikož design vyvíjené hry byl postavený na myšlence držení a vrhání disků, bylo mým hlavním cíle pochopit jak realizovat uchopení a upuštění objektů. V ukázkových scénách zaměřených na interakce s objekty jsem identifikoval komponenty sloužící k vytvoření jednoduchého úchopu objektů. Pro rozlišení objektů, se kterými hráč může interagovat je využita třída *VRTK_InteractableObject*. Ta poskytuje nejrůznější možnosti nastavení, například zdali je objekt uchopitelný a kterou rukou jej lze uchopit. Samotné uchopení lze poté realizovat ve vícero variantách. Pro pevné uchopení objektu slouží třída *VRTK_FixedJointGrabAttach* a lze u ní určit, jak pevně jsou objekty v ruce drženy.

S těmito znalostmi jsem vytvořil jednoduchou scénu pro otestování správného fungování těchto prvků. Do této scény bylo zapotřebí vložit několik dalších objektů obsahující skripty pro správné fungování virtuální reality. Tyto skripty inicializují a spravují komunikaci se samotným zařízením a vytváří reprezentaci hráče ve scéně. Od grafiků jsem obdržel jednoduché modely pro disky a level, které jsem do této scény vložil a otestoval, zdali lze disky uchopit a poté upustit.

V neposlední řadě bylo třeba vyřešit možnost pohybu ve scéně. Ačkoli design vyvíjené hry byl založený na statickém umístění hráče v dané aréně, tato potřeba vyplývala z nutnosti umožnit grafikům pohyb po scéně, tak aby ve virtuální realitě mohli zkontrolovat správnost svých modelů a jejich umístění.

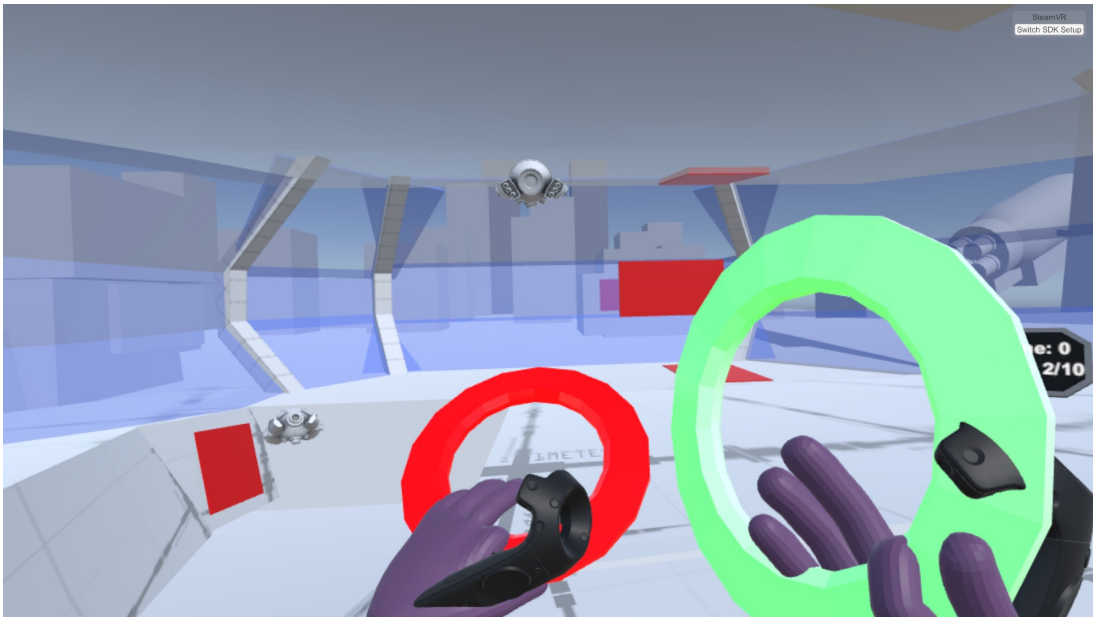
Ačkoli je pohyb brán jako základ většiny her, virtuální realita tuto činnost dosti komplikuje. Jelikož má hráč fyzicky omezený prostor, ve kterém se může pohybovat, nelze využít pouze jeho sledování v tomto prostoru. Většinou je tedy zapotřebí pohyb hráče zprostředkovat skrze určitou mechaniku, která jeho pohyb nahradí. Realizace ne-fyzického pohybu hráče má ovšem svá omezení a rizika.

Užití nevhodné metody přesunu hráče může u něj vyvolat kinetózu. Jedná se o nepříjemný stav, který se u lidí většinou vyskytuje při přesunu motorovým vozidlem nebo lodí. Jeho důsledkem je nevolnost jedince vedoucí případně až ke zvracení. Příčinou kinetózy je obdržení konfliktních signálů z očí a vestibulárního systému (smyslový orgán v uchu zodpovědný za prostorovou orientaci a rovnováhu).[33][34] Pokud tedy hráč nehybně stojí a dívá se rovně před sebe, ale ve stejný moment se jeho pohled pohybuje přesunem kamery, mozek tohoto hráče zároveň

obdrží informaci o stání a pohybu. Tento konflikt signálů u hráče vyvolá kinetózu a s ní spojenou nevolnost.[35]

K eliminaci tohoto nežádoucího efektu se využívají nejrůznější metody určené k oklamání lidských smyslů.[36][37] Jelikož ve hře samotné neměl být pohyb využíván, nebylo nutné použít ty nejlepší a nejkomplexnější metody. Cílem bylo tedy najít co nejméně časově a implementačně náročnou variantu. Jelikož knihovna VRTK obsahuje řešení za využití takzvané Fade/Blink metody, rozhodl jsem se ji využít. Opět jsem otestoval tuto funkcionalitu v ukázkových scénách, identifikoval komponenty, které ji realizují a vložil ji do svojí testovací scény.

Základem této metody je využití teleportace hráče, která je mu skryta. Hráč nejprve za využití ukazatele určí pozici, na kterou se chce přesunout. Pokud je zvolená pozice validní, je hráči na moment zakryt jeho výhled vložením černé plochy do bezprostřední blízkosti kamery. Následně je hráč přemístěn na zvolené místo a jeho výhled je posléze obnoven. [35] Tento přístup může být upraven, kdy namísto kompletního zakrytí výhledu hráče jsou před něj v rychlé sekvenci vloženy a odebrány dvě černé plochy simulující mrknutí hráče. Jelikož lidský mozek vjem mrknutí přirozeně vytlačuje, je mu proces teleportace lépe skryt.[38]



Obrázek 2: Hra po několika týdnech vývoje

6.2 Implementace základní funkcionality disků

V rámci první iterace bylo mým dalším úkolem implementovat základní chování disků. To se skládá z několika částí: let disků po aréně, odrazy disků od stěn arény, jejich natočení vůči těmto stěnám během letu a návrat disků do rukou hráče.

Jako první bylo třeba vyřešil pohyb disku. Game designér mi vysvětlil, že by ve hře nedávalo smysl mít realistické házení diskem. V takovém případě by bylo pro hráče náročné vrhem kompenzovat vůči gravitaci působící na disk a navíc by jeho některé další vlastnosti, jako například odrazy od stěn, nebyly realizovatelné. Mělo se tedy jednat o přímočarý let, kdy na disk nepůsobí gravitace ani žádné další vnější síly.

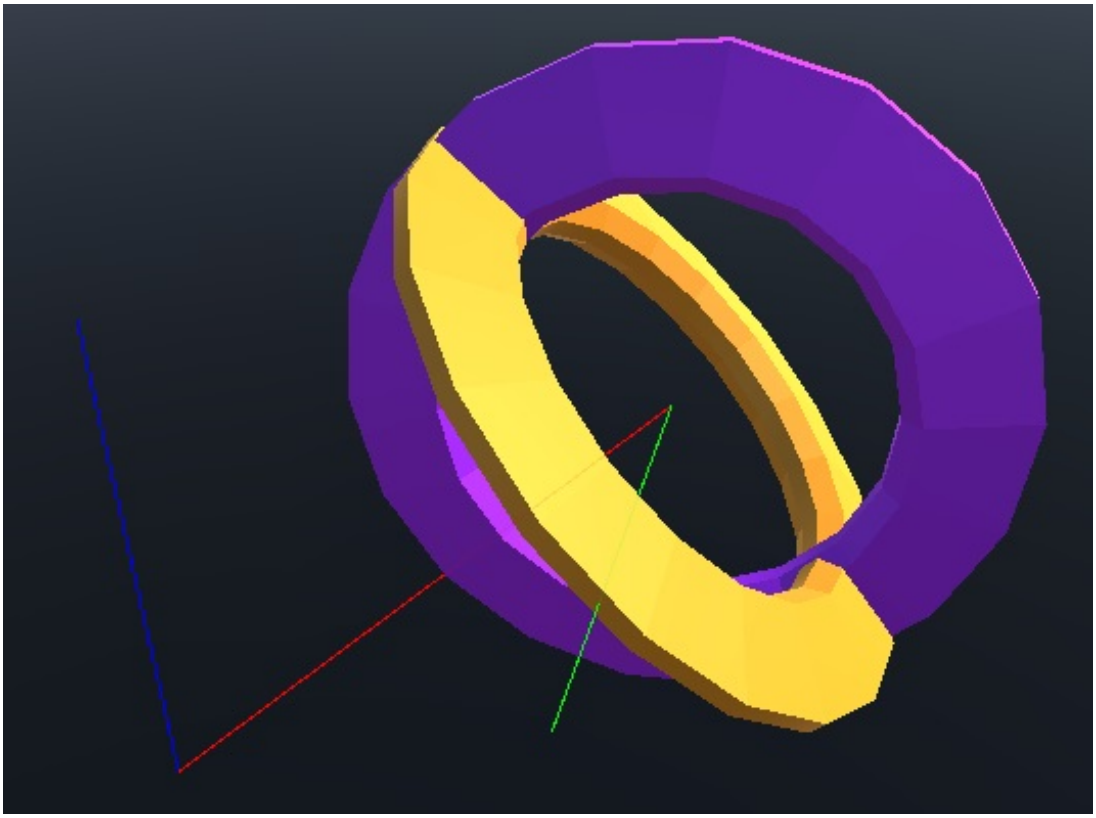
Pro tyto účely se využila událost informující o upuštění objektu hráčem, která je vyvolána instancí třídy *VRTK_InteractableObject* umístěnou na discích. Reakcí na tuto událost je aktivován skript, který od fyzikálního enginu získá aktuální směr pohybu daný vrhem a posléze mu určuje konstantní sílu, kterou se má disk v tomto směru pohybovat. Velikost této síly lze nastavit v inspektoru a tímto určit rychlost pohybu disku.

Později byl identifikován problém, kdy pokud hráč držel ruku na místě a pokusil se vypustit disk, byl tento disk vystřelen ve víceméně náhodném směru. Toto bylo způsobeno přesností senzorů, zachycující i minimální pohyby ruky hráče. Tento problém byl vyřešen přidáním kontroly rychlosti pohybu disku při jeho upuštění. Pokud jeho rychlost nedosahovala zvolené hodnoty, byl místo vystřelení pouze upuštěn na zem.

Následně jsem se zaměřil na rotaci disků za letu. Jejím cílem bylo zajistit to, aby disky vždy letěly kolmo vůči stěnám arény. Pokud by disk letěl souběžně se stěnou, do které narazí, tento odraz by vypadal nepřirozeně. Pro řešení tohoto problému je při každé fyzikální aktualizaci hry za letu disku prováděn následující výpočet.

Je získán aktuální směr letu disku (na obrázku zvýrazněn červenou čarou) a pomocí raycastu je získán bod stěny, vůči které disk letí. V tomto bodě je získána normála (znázorněna modrou čarou) a pomocí cross produktu této normály a vektoru směru letu disku je vypočítán vektor reprezentující směr vzhůru (znázorněn zelenou čarou), který disk musí mít, aby letěl kolmo vůči této stěně. Výsledná rotace disku (reprezentována diskem ve fialové barvě) je získána pomocí funkce *Quaternion.LookRotation()*, které jsou předány vektory směru pohybu disku a vypočítaného směru vzhůru. Disk je posléze natočen pomocí funkce *Quaternion.RotateTowards()*, která otočí diskem ze současné rotace k rotaci finální o maximální počet stupňů. Omezením této rotace v závislosti na čase vytvoří efekt postupného natočení disku vůči stěně.

Jako další byly implementovány odrazy disků od stěn arény. Mělo se jednat o jednoduché odrazy, kdy úhel dopadu se rovná úhlu odrazu. Zde byl výpočet o něco jednodušší a daná logika je následovná. Při detekci kolize disku se stěnou arény je získána normála z plochy, se kterou disk kolidoval a směr letu disku před touto kolizí. Tyto dva vektory jsou poté využity ve funkci *Vector3.Reflect()*, která vrátí nový směr letu disku.



Obrázek 3: Vizualizace výpočtu rotace disku

Jako poslední byl řešen návrat disků do rukou hráče. Po zachycení události o stisknutí zvoleného tlačítka ovladače je vypnut skript pro normální pohyb disku a místo něj je spuštěn skript pro jeho návrat. Ten z aktuální pozice disku a ovladače, ke kterému přísluší, určí směr letu takový, aby disk letěl hráči přímo do ruky. V momentě kdy se disk dostane do blízkosti ovladače je instancí třídy *VRTK_InteractableObject* vyvolána událost, na kterou je reagováno explicitním zavoláním funkce pro uchopení objektu na instanci třídy *VRTK_InteractGrab*. Po uchopení disku jsou na něm vypnuty veškeré skripty spojené s pohybem. Pro návrat disku je možné nastavit rychlost letu odlišnou od rychlosti pro klasický let. Tímto lze návrat disku učinit rychlejším, tak aby hráč na disk nemusel příliš dlouho čekat.

Později byla k návratu disku přidána časomíra pro případ, kdyby se disk zasekl o nějakou překážku. Pokud není disk do daného časového intervalu uchycen, je teleportován na pozici svého příslušejícího ovladače a je zavolána dříve zmíněná funkce pro jeho uchycení.

Další problém se vyskytl při zvolení příliš vysoké rychlosti pro návrat. Mohlo nastat, že se během jednoho snímku disk posunul až za ruku hráče a v tomto případě nebylo možné uskutečnit jeho uchycení. Do funkce pro návrat disku byl přidán výpočet aktuální vzdálenosti mezi diskem a ovladačem a vzdálenosti, kterou disk urazí během jedné fyzikální aktualizace hry. V případě kdy je aktuální vzdálenost menší než ta, kterou disk urazí, je aplikován stejný postup jako u teleportace disku.

6.3 Projektily, atributy hráče a nepřátel a jejich úmrtí

S dokončeným základem pohybu disků byly mé další úkoly zaměřené na boj hráče s nepřáteli. Pro tyto účely bylo nutné vytvořit systém atributů, ve kterém by jednotlivé entity měly uchovány své vlastnosti jako například aktuální stav životů.

Nejdříve bylo potřeba definovat třídu *Attribute*, která představuje jeden obecný atribut. V ní se definuje aktuální a maximální hodnota, kterou může daný atribut nabývat. Dále má třída definovanou událost informující o změně aktuální hodnoty atributu. Ta je využívána hlavně pro reakci na změnu životů hráče a nepřátel. Například při klesnutí životů pod jedna dojde k vyvolání specifické akce pro smrt (zvýšení skóre, konec hry) nebo při zranění hráče je aktualizováno UI informující o jeho současném stavu životů.

Jednotlivé instance atributů dané entity spravuje instance třídy *CharacterAttributes*. V případě kdy skript reagující na zasažení projektilem chce snížit hráči životy, získá od této třídy referenci na atribut spravující životy a změní jeho hodnotu podle množství zranění, které projektil nese. Tato hodnota je udržována v instanci třídy *DamageModule*, která je na discích a projektilech.

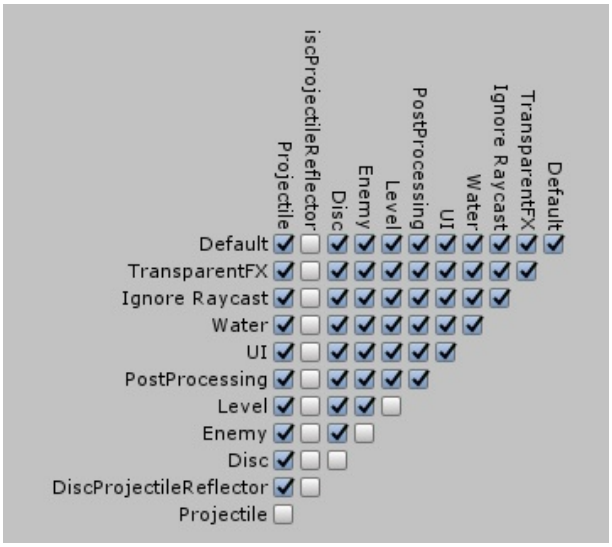
Samotné projektily fungují jako prefab, předloha herního objektu, jehož kopie lze za běhu hry vytvářet. V momentě, kdy chce nepřítel po hráči vystřelit, vytvoří instanci tohoto prefabu a určí mu směr jeho letu. Pomocí tagů je identifikována kolize s hráčem a dříve popsaným způsobem mu jsou ubrány životy. Při kolizi projektilu s diskem dojde k jeho odrazu a je na něm aktivována možnost zranění nepřátel. Takto odražené projektily mohou zranit i ostatní nepřátele, kterým projektil původně žádné poškození nečinil.

Pro řešení úmrtí byla vytvořena abstraktní třída *DeathHandler* s metodou *HandleDeath()*, z které jednotlivé verze úmrtí dědí a tuto metodu implementují. *EnemyDeathHandler* přistupuje k čítači skóre a zvýší jeho stav o hodnotu bodů nepřítele. *PlayerDeathHandler* pracuje s instancí třídy *GameEnd*. Ta se stará o zapsání skóre do databáze a navrácení hráče do menu.

Při vložení nepřátel a projektilů do hry bylo potřeba učinit rozhodnutí, které z objektů ve scéně mají vystupovat jako pevné collidery a které jako průchozí triggery. Collidery představují fyzikální reprezentaci objektu a jsou využívány pro výpočet kolizí mezi objekty. Triggery jsou collidery nastavené jako trigger a fyzikální engine je bere jako průchozí. Jsou tedy vhodné pro objekty, u kterých nechceme, aby kolidovaly s ostatními, ale bylo u nich možné získat informaci o jejich vzájemném dotyku.

U prvotních implementací jsem opakovaně narážel na problém, kdy u některých objektů bylo zapotřebí chování jak colliderů tak triggerů. Například u projektilu je nutné, aby kolidoval s disky, ale ne s ostatními projektily. K řešení toho problému bylo vhodné využít vrstev fyzikálního enginu. Tyto vrstvy lze volně přidávat a u každé vrstvy lze nastavit, s kterými vrstvami má povoleno interagovat. U každého objektu ve scéně lze poté vybrat, do které vrstvy spadá a tímto určit, s kterými objekty bude kolidovat.

Ve výsledném řešení bylo určeno, že hráč a nepřátelé budou reprezentováni pomocí triggerů, tak aby jimi mohly disky a projektily volně procházet, ale o této interakci byly informovány patřičné skripty. Level, disky a projektily využívají colliderů a jsou odděleny do odlišných vrstev. Vrstvy pro disky a projektily jsou nastaveny tak, že jejich objekty nemohou kolidovat s ostatními objekty ze stejné vrstvy.



Obrázek 4: Nastavení fyzikálních vrstev

6.4 Opravy nechtěných interakcí a vyladění základních mechanik

Během testování dříve popsaných mechanik bylo objeveno několik nežádoucích interakcí. Jelikož správná funkčnost daných mechanik byla prioritou, nebyla jim do tohoto bodu věnována pozornost. Protože jádro funkcionality hráče bylo nyní naimplementováno, zaměřil jsem se na eliminování těchto nechtěných interakcí. Jmenovitě se jednalo o případy, kdy při odrazu projektilu diskem došlo k vyhození daného disku z hráčovy ruky, dále při pohybu hráče jeho pohled občasně problikával a tento stejný jev se zřídka vyskytoval při zasažení hráče projektilem. Rozhodli jsme se v rámci těchto oprav zároveň přepracovat některé z dalších systémů hry, tak aby byly lépe připraveny na funkcionalitu, která měla být do hry přidána v budoucnu. Z tohoto důvodu byla většina níže popsané práce vykonána za využití párového programování.

Nejjednodušší na vyřešení bylo vyrážení disků z rukou hráče při jejich zasažení projektilem. Na skriptu *VRTK_InteractableObject* je parametr určující množství energie, jejíž převýšení při kolizi způsobí upuštění objektu. Nastavením tohoto parametru na hodnotu, které projektily nemohly při kolizi dosáhnout, zamezilo vzniku tohoto jevu.

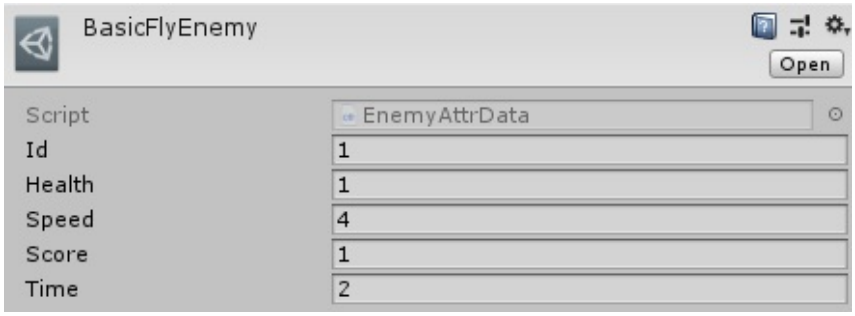
Po bližším prozkoumání problikávání výhledu hráče při jeho pohybu a zásahu projektilem bylo objeveno, že dochází k jeho teleportování. Dalším testováním a analýzou patřičných skriptů se došlo závěru, že skript určený k umístění kamery podle výšky uživatele opakovaně prováděl svoji funkcionalitu při pohybu hráče. Zároveň při tomuto teleportování docházelo k přepínání

collideru hráče z triggeru na běžný collider. Pokud v daný moment byl hráč zasažen projektilem, došlo k jejich vzájemné kolizi vedoucí k posunu hráče a jeho dalšímu teleportování. Jediným řešením tohoto problému bylo vypnout daný skript a tímto mu zamezit ve tvorbě těchto nechtěných interakcí.

Vypnutím tohoto skriptu však vznikl problém, kdy kamera nebyla umístěna do správné výšky. Tímto většinou u hráče vznikal vjem, že je výše než by měl být a tudíž levituje nad podlahou arény. Jelikož z dříve zmíněných důvodů nemohl být daný skript ponechán zapnutý, bylo zvoleno alternativní řešení. Funkcionalita tohoto skriptu byla využívána jiným skriptem reprezentujícím fyzikální chování těla hráče. Byla vytvořena třída dědící z tohoto skriptu, kde byla přetížena metoda *Start()*. Toto je metoda vyvolávaná na všech instancích tříd dědících z třídy MonoBehaviour (jedná se o základní třídu, která se vyskytuje na herních objektech ve scéně), která je volána před první aktualizací hry. V této metodě je objeven nejbližší objekt levelu arény, který se nachází přímo pod hráčem. Poté je explicitně zavolána metoda pro teleport, které je předán zasažený bod plochy nalezeného objektu. Je využit dříve zmíněný problematický skript, pomocí kterého je kamera přemístěna do správné výšky a posléze je tento skript vypnut, tak aby nezpůsoboval nežádoucí teleportace hráče.

V rámci párového programování jsem pomohl druhému programátorovi v přepracování systému pro vytváření nepřátel ve scéně. Tento systém v původní verzi umožňoval pouze vytvářet jednu konfiguraci daného nepřítele pro danou arénu, která byla společná pro všechny herní módy. Jelikož bylo žádoucí mít různé varianty nepřátel pro různé módy, bylo potřeba tento systém přepracovat.

V nové verzi tohoto systému bylo využito třídy *ScriptableObject*, která je alternativou k třídě *MonoBehaviour*, a je často využívána pro ukládání různých konfigurací dat. Byla tedy vytvořena třída udržující data pro daný typ nepřítele, obsahující jeho životy, bodové ohodnocení atd. V tomto novém spawn systému je poté možné pro daný level a herní mód vložit zvolenou konfiguraci daného nepřítele, kde při vytvoření nové instance tohoto nepřítele jsou tato data na ní aplikována.



Obrázek 5: Konfigurace nepřítele

6.5 Přidání doměřování disků

Jako poslední hlavní vlastnost disků, kterou bylo potřeba implementovat, byla jejich schopnost se navádět na blízké nepřátele. Tato nutnost vyplývala z potřeby pomoci hráči zasáhnout nepřátele v případech, kdy by minimální nepřesnost hodu vedla k jejich minutí. V tomto případě by se mohl hráč mylně domnívat, že disk hodil přesně. Pokud by se tato situace vícekrát opakovala, mohl by u hráče vzniknout pocit frustrace vedoucí k jeho ukončení hry.

Drobná nepřesnost hodů je způsobená potřebou aktivovat vstup na ovladači pro vypuštění disku, jenž v reálném světě není nutný. Docházelo tedy k tomu, že hráč aktivoval tento vstup dříve nebo později, než chtěl disk ve skutečnosti hodit. Toto způsobilo, že vektor letu disku měl mírně odlišný směr a ačkoli tato odchylka nebyla veliká, byla dostačující proto, aby disk nezasáhl určený cíl.

Schopnost doměřit se na cíl měla za účel kompenzovat vůči těmto odchylkám, avšak tak aby u disku nebyly silně znatelné změny směru letu a tudíž hráč nezaznamenal zásah tohoto systému do jeho hodu. Jako první bylo nutné stanovit pravidla, podle kterých se doměřování disků mělo řídit:

1. Pokud se ve směru letu disku nenacházejí žádní nepřátelé, tak tento směr nesmí být pozměněn.

2. V případě, že je detekován potencionální cíl, avšak jeho vzdálenost od disku je příliš malá nebo veliká, je tento cíl ignorován. Tímto je zamezeno ostrému zatočení disku při malé vzdálenosti a velikému odchýlení od původního směru letu u dlouhé vzdálenosti.

3. V situaci, kdy se ve směru letu disku vyskytuje více než jeden nepřítel, bude disk doměřen na nepřítele, jehož odchylka od současného směru letu je nejnižší. Tímto je změna směru letu co nejmenší a tudíž i méně zaznamenatelná.

4. Pokud je disk v dostatečné blízkosti svého aktuálního cíle a je podle pravidla popsaného v minulém bodě identifikován nový cíl, je teto pravidlo ignorováno a cíl disku nezměněn. Tímto je zabráněno vícenásobným změnám směru letu disku způsobeným neustálým pohybem nepřátel.

5. V případě, kdy více nepřátel má stejnou odchylku od disku, je zvolen jako cíl nepřítel, jehož vzdálenost od disku je nejmenší.

Při samotné implementaci bylo nejprve nutné vyřešit schopnost detekovat nepřátele ve směru letu disku. Jako první variantou se nabídlo využít spherecastu, verze raycastu, u které je pro detekci objektů namísto paprsku využita koule o zvoleném poloměru. Jelikož se však jedná o výpočetně náročnou operaci [39], jejíž časté použití by mohlo vést k výraznému zpomalení běhu hry, rozhodl jsem se ji nakonec nevyužít.

Jako zdaleka lepší řešení se nabídlo vytvoření systému, který by v sobě uchovával informace o pozicích všech aktivních nepřátel. Z informace o pozici nepřítele lze zjistit jeho vzdálenost od

disku a porovnat směr letu disku se směrem k tomuto nepříteli. Tímto se proces identifikace cíle stává mnohem méně výpočetně náročným. Vytvořil jsem tedy správce, jenž měl tyto informace uchovávat a požádal jsem svého kolegu, aby do svého systému pro vytváření nepřátel přidal registraci reference na jejich *Transform* komponentu do tohoto správce. Taktéž jsem do skriptu pro úmrtí nepřátel přidal jejich odregistrování.

Ve výsledné implementaci se skript pro pohyb disku vždy dotáže skriptu pro doměřování, zdali má nějaký cíl. Ten získá reference na pozice všech aktivních nepřátel a z nich vybere ty, kteří jsou ve vhodném úhlu a vzdálenosti od disku. Pokud tato funkce vrátí více než jednoho nepřítele, je provedeno jejich porovnání podle dříve definovaných pravidel a je zvolen jeden nepřítel jako cíl. Poté si skript pro pohyb disku od skriptu pro doměřování získá nový směr letu.

```
private TargetData[] GetPossibleTargets(List<Transform> enemies)
{
    List<TargetData> posTargets = new List<TargetData>();
    Vector3 pos = _transform.position;
    Vector3 dir = _rigidbody.velocity.normalized;

    foreach (Transform enemy in enemies)
    {
        Vector3 enemyPos = enemy.position;
        float distance = Vector3.Distance(pos, enemyPos);
        if (distance < _minTrackingDistance || distance > _maxTrackingDistance)
            continue;

        Vector3 toEnemy = (enemyPos - pos).normalized;
        float angle = Vector3.Angle(dir, toEnemy);
        if (angle > _trackingAngle)
            continue;

        TargetData td = new TargetData(enemy, distance, angle);
        posTargets.Add(td);
    }

    return posTargets.ToArray();
}
```

Výpis 1: Určení nepřátel ve směru letu disku

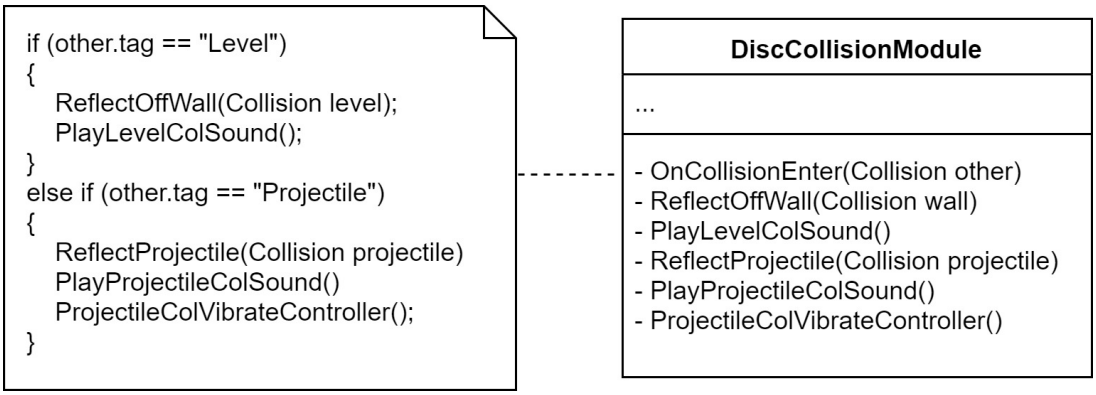
6.6 Refaktorizace disků a s ní související přístup k architektuře

Do tohoto bodu bylo naším hlavním cílem porozumět a naučit se používat engine Unity, soustředil jsem se proto primárně na funkčnost hry. Jelikož jsem doposud nikdy nepracoval na takto rozsáhlém projektu, neměl jsem dostatečné zkušenosti, které by mi během vývoje umožnily lépe dopředu plánovat architekturu kódu hry. Důsledkem toho byla skutečnost, že většina prvků hry byla těžko rozšiřitelná a náchylná na změny.

Rozhodl jsem se tedy pokud možno co nejvíce vyvarovat případným budoucím komplikacím plynoucím ze změn požadavků aplikováním lepších přístupů k architektuře. Po konzultaci s ostatními programátory ve firmě mi bylo doporučeno vytvářet jednotlivé skripty a herní objekty za využití komponentní architektury a řešit jejich vzájemnou komunikaci a řízení pomocí událostmi řízené architektury.

Hlavní myšlenkou využití komponentní architektury při vývoji her v enginu Unity je rozdělení větších prvků na více menších, kdy každý jednotlivý prvek reprezentuje pouze jednu danou funkcionalitu.[40][41] Ve hře se může například vyskytovat nepřítel, který se hýbe po mapě, detekuje hráče a střílí po něm. Místo vytvoření jediné třídy řešící jeho veškeré chování, lze tuto třídu rozdělit na vícero menších tříd. Celkové chování nepřítele je pak definováno souhrnem komponent, ze kterých je tvořen. Jednotlivé komponenty pak lze přidávat a odebírat a tímto jeho chování měnit. Můžeme mu například přidat komponentu umožňující mu se zneviditelnit nebo vyměnit jeho komponentu pro pohyb za jinou variantu s odlišnou logikou.

Ačkoli sám engine Unity navádí k využití komponentní architektury (všechny herní objekty jsou tvořeny komponentami), tento přístup jsem nevyužíval dostatečně. Jako příklad bude využit skript pro kolize disku, v jehož původní verzi byla implementována veškerá funkcionalita spojená s kolizemi disků s arénou a projektily. Pokud bylo potřeba u těchto kolizí upravit nebo přidat novou funkcionalitu, bylo nutné zasahovat do kódu tohoto skriptu.



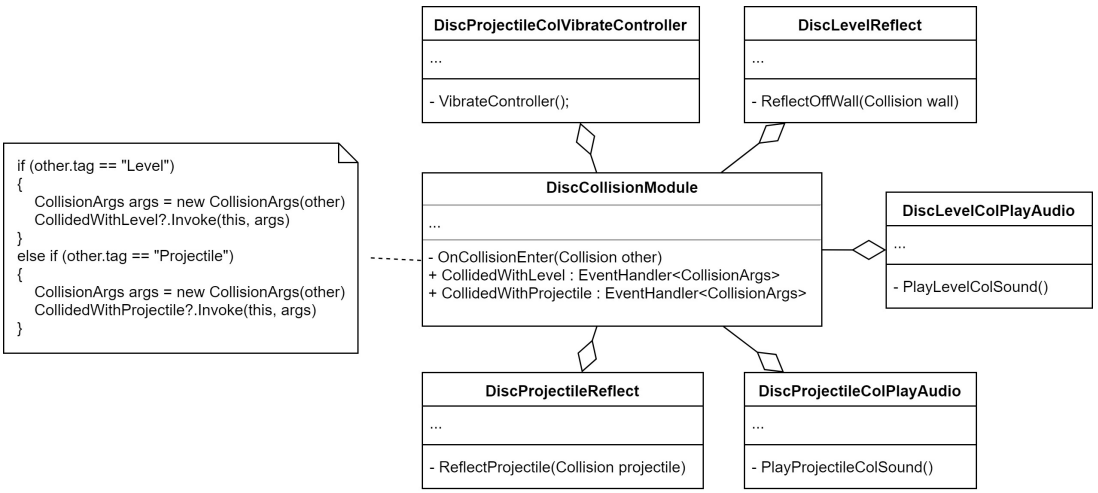
Obrázek 6: Původní verze kolizního skriptu pro disky

Aplikováním komponentní architektury by tento skript šlo rozdělit na vícero menších skriptů, obsahující jednotlivé části jeho původní funkcionality. Samotný kolizní skript by v tomto případě pouze vyvolával metody ostatních komponent v závislosti na objektu, se kterým kolidoval. Pokud by bylo potřeba pozměnit chování jednotlivých části funkcionality kolize disku, byla by tato změna provedena pouze v patřičném skriptu určeném pro tuto vlastnost.

Nedostatkem tohoto řešení je skutečnost, že jakékoliv přidání, odebrání anebo změna komponent na daném herním objektu stále vyžaduje zásah do kódu daného skriptu. Pokud by na disku byla odebrána komponenta pro spuštění zvukového efektu při kolizi s projektilem a místo něj přidána například komponenta s particle efektem, nebyl by tento nový efekt spuštěn a došlo by k vzniku chyb z důvodu chybějící reference na starou komponentu.

Tento problém lze vyřešit využitím událostmi řízené architektury. Zde dochází k převrácení závislostí mezi jednotlivými komponentami a vyvolávání jejich individuální funkcionality za využití událostí. Skript pro kolize v tomto případě definuje události informující o kolizích s levelem a projektily. Jednotlivé komponenty se poté na tyto události registrují a reakcí na jejich vyvolání provedou svoji danou funkcionalitu.

Jakákoliv změna kompozice komponent v tomto momentě už nemá žádný vliv na dříve vytvořený kód. U nových komponent se stačí registrování na vhodnou událost v jiné komponentě a u odebrání komponent nevznikají chybějící reference.



Obrázek 7: Přepracovaný kolizní skript

Po pochopení těchto přístupů jsem se je rozhodl aplikovat na disky, jakožto nejvýznamnější část hry, na které jsem doposud pracoval. Před přepracováním disků jsem si vytvořil třídní diagram pro veškeré komponenty, ze kterých by disk měl být tvořen a na základě tohoto diagramu jsem vytvořil zcela novou verzi disků. Tímto byly disky rozšířeny z původních šesti skriptů na necelou dvacítku.

6.7 Nahrazení disků štíty a vrhanými koulemi

Při průběžných prezentacích hry zaměestnancům firmy jsme opakovaně naráželi na problém, kdy někteří hráči v porovnání s jinými měli výrazně větší problém hodit disk v jimi zamýšleném směru. Odchylky jejich hodů byly dokonce natolik veliké, že nešly ani kompenzovat doměřováním bez jeho viditelného zásahu do letu disku.

Po bližším prozkoumání jsme identifikovali společnou vlastnost těchto hráčů, kterou byl jejich odlišný styl hodu disku. Zatímco většina hráčů házela disky přes své rameno, jejich způsob byl podobný hodu klasického létajícího talíře. Jelikož se jednalo o skutečnost, která by silně znehodnocovala zážitek ze hry, a dalo se předpokládat, že by se určité procento hráčů snažilo tímto stylem disky házet, bylo zapotřebí tento problém eliminovat.

Jelikož jsme potřebovali být schopni změřit jak silný vliv tento problém má a jestli naše snahy o jeho řešení mají pozitivní efekt, rozhodl jsem se vytvořit speciální tréninkovou scénu. V této scéně byl umístěn statický cíl, u kterého bylo možné měnit jeho vzdálenost a velikost. Dále tato scéna obsahovala čítač hodů a zásahů, který tyto informace společně s procentem úspěšnosti hodů zobrazoval hráči.

Společně s těmito určitými hráči jsme potom zkoušeli házet na cíl, abychom zjistili, jaké procento hodů tímto stylem mine cíl. Nasbíraná data ukázala, že při této metodě hodu nebyl nikdo schopný dosáhnout ani třetinové úspěšnosti. V porovnání s přibližně osmdesáti procentní úspěšností zásahů při hodu vrchem byla tato informace alarmující. Hrozilo totiž, že by hráči házející tímto stylem považovali hru za rozbitou a v souladu s podmínkami služby Steam by požadovali o vrácení produktu.[42]

Jako první příčinu tohoto problému jsme považovali způsob držení disku. Jelikož jsme měli disky v rukou hráče umístěny tak, aby vizuálně seděly s držením ovladače při hodu vrchem, jejich umístění při hodu talířem nebylo adekvátní. Změnili jsme tedy pozice a rotace disků, tak aby více odpovídaly držení talíře. Následné testování překvapivě ukázalo, že tato změna měla téměř nulový efekt na úspěšnost hodů. Jako vedlejší efekt se projevil fakt, že každý hráč při tomto stylu hodu držel ovladače odlišně a bylo nutné každému individuálně disky přenastavit. Toto by samozřejmě ve hře samotné nebylo možné uskutečnit, a proto jsme tuto cestu řešení našeho problému označili jako mylnou.

Po dalším testování, při kterém jsme využili i skutečného létajícího talíře, jsme nakonec identifikovali jako jádro problému skutečnost, že při hodu talířem jej člověk vyšle v jiném směru než je jeho cíl. Toto je vyrovnáno rotací, kterou na talíř aplikuje svými prsty a tento jev lze pozorovat na jeho letu po křivce. Aplikováno na naši hru, při vrhu tímto stylem hráči sami házeli disk nepřesně, a jelikož nebylo možné emulovat zmíněnou zpětnou rotaci, nešlo tento problém odstranit. Bylo tedy nutné se poohlédnout po alternativách k diskům, u kterých by se tento problém nevyskytoval.

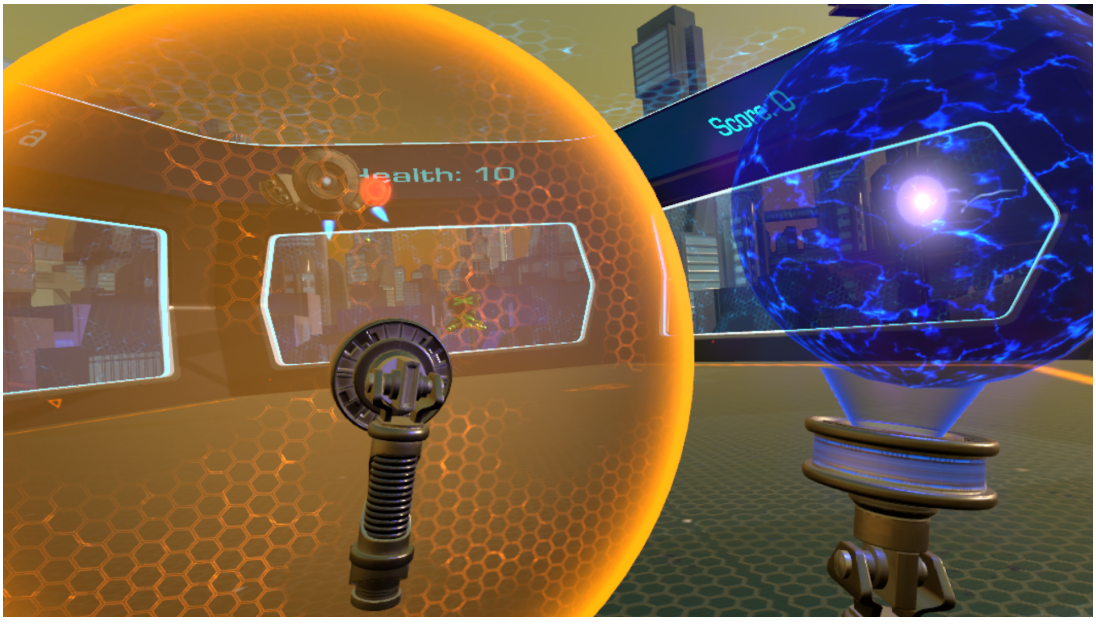
Ve hrách pro virtuální realitu se vyskytují nejrůznější druhy zbraní. Jedny z těch nejčastěji využívaných jsou střelné a sečné zbraně. Jejich popularita je dána nenáročnou implementací,

jelikož jsou tyto zbraně vždy pevně umístěny v rukou hráče. Protože jsme však chtěli zachovat mechaniku vrhu předmětů, nebylo možné těchto variant využít. Vyzkoušeli jsme tedy vícero herních titulů dostupných na trhu a nakonec jsme se inspirovali hrami Sparc a Blue Effect a jejich implementací vrhaných koulí.

Začal jsem tedy pracovat na nahrazení disků těmito koulemi. Jelikož při hodu koulí v reálném životě má člověk vždy rozevřenou dlaň, mohlo by se stát, že při hodu koulí ve hře by hráči instinktivně upustili svůj ovladač. Rozhodli jsme se proto hráči do rukou vložit tyče reprezentující ovladače, na které jsou koule samotné umístěny. Tyto tyče jsou hráči do rukou vloženy stejným způsobem jako dříve popsané sečné zbraně a u koulí je při jejich uchycení aplikován offset jejich pozice, tak aby byly nad nimi vhodně umístěny.

Díky předešlému přepracování disků bylo možné využít většinu jejich komponent i u koulí. Případné rozdíly byly řešeny vytvořením zcela nových komponent nebo alternativních variant komponent disků. Například byla vytvořena nová komponenta pro vypnutí kolize koule s projektily při jejím držení v ruce hráče. Jelikož by tento jev nevypadal přirozeně, byly pro účely odrazů projektilů vytvořeny štíty. Pokud hráč v dané ruce drží kouli, lze ji přepnout na štít a z něj zpět na kouli.

Jelikož jsme chtěli ve hře udržet dynamiku házení jak levou tak i pravou rukou, rozhodli jsme se omezit množství zásahů, které štít může vyblokovat. Při každém odražení projektilu je štítu odebráno určité množství jeho energie, čímž je štít zmenšen. Při nízké energii štítu je už obtížné jím projektily blokovat a hráč je tudíž nucen použít své ruce pro opačné účely. Na energii štítu je poté aplikována pasivní regenerace, která štít obnovuje i při jeho nepoužívání. Pokud tedy hráč potřebuje opět vyměnit pozici koule a štítu, je mu tento dříve vyčerpaný štít znovu dostupný.



Obrázek 8: Štít a vrhaná koule

6.8 Finální ladění a balancování hry

V posledních několika dnech na praxi jsme se plně věnovali posledním úpravám hry a ladění jejich jednotlivých prvků. Požádali jsme všechny zaměstnance firmy, aby hru otestovali a dali nám zpětnou vazbu k věcem, které by bylo vhodné pozměnit. Z těchto informací jsme vybrali ty změny, které jsme byli schopni ve zbylém času implementovat.

Bylo například pozměněno UI informující hráče o jeho počtu životů, tak aby bylo ve scéně více viditelné. Dále byl přidán zvukový efekt zvýrazňující jeho nízký počet životů. Byl také pozměněn herní mód *Time Attack*, ve kterém byly hráči odebrány životy. Namísto toho každý jeho zásah projektilem mu odebral určité množství zbývajícího času.

Nejvýraznější změny mnou implementovaných mechanik byly úpravy hodů koulí. Původně hráči koule upadla na zem v případě, kdy ji držel na místě a pokusil se ji hodit. Tato logika byla přepracována, tak aby koule zůstala hráči v ruce jejím okamžitým znovu uchycením. Taktéž byla zvýšena hodnota, podle které bylo určeno, zdali byla koule skutečně hozena. Tímto je hráč nucen kouli hodit silněji, za což je odměněn jejím rychlejším letem. Taktéž byla na vrácení koulí do rukou hráče přidána vibrace ovladačů pro zvýšení vjemu tohoto jevu.

V neposlední řadě jsem vylepšil systém pro animaci přechodu mezi koulemi a štíty, tak aby bylo možné vytvořit vícero jejich variant. Podle předloh od grafiků v týmu jsem jednotlivé animace implementoval, z niž nakonec byla jedna určena jako nevhodnější a trvale aplikována.

Obrázek 9: Testování hry

7 Uplatněné teoretické a praktické znalosti ze studia

Během bakalářské praxe jsem využil znalostí z mnoha předmětů absolvovaných během studia. Základem byla znalost objektově orientovaného programování z předmětů *Programování I* a *Programování II*, rozšířena o znalost jazyka C# z předmětů *Programovací jazyky II* a *Architektura technologie .NET*. Dále bylo potřeba umět navrhnout a implementovat složitější algoritmy, čemuž byly věnovány předměty *Algoritmy I* a *Algoritmy II*. Při své práci jsem využil znalostí tvorby UML diagramů a využití návrhových vzorů, taktéž jsem prakticky aplikoval teorii softwarového vývoje, jimž jsem se naučil v předmětech *Úvod do softwarového inženýrství* a *Vývoj informačních systémů*. Ačkoli jsem v rámci projektu prakticky s renderováním nepracoval, mohl jsem porovnat jeho realizaci v enginu Unity se svým projektem z předmětu *Základy počítačové grafiky*. V neposlední řadě jsem měl možnost seznámit se s technologie a metodikami, které grafici v týmu využívali pro tvorbu svých modelů a porovnat je s těmi probranými v předmětu *Modelování v grafických aplikacích*.

7.1 Scházející teoretické a praktické znalosti

Jelikož mnou studovaný obor nenabízí předměty zaměřené na tvorbu her, využil jsem pouze zkušeností z tvorby jednodušších her v rámci projektů v několika málo předmětech. Většinu svých znalostí týkajících se tohoto odvětví a herního enginu Unity jsem před absolvováním praxe tedy získal z osobního zájmu mimostudijně.

Novou zkušeností byla také práce v týmu obsahující jak členy se stejným tak i odlišným studijním zaměřením. Protože žádný mnou absolvovaný předmět neobsahoval týmový projekt, ve kterém by byl společně vytvářen softwarový produkt, bylo nutné si zvyknout na nové elementy vstupující do vývoje. Toto obsahuje rozdělení úkolů mezi jednotlivé členy, diskuze a porady spojené s vývojem, vzájemná výpomoc při případných problémech a další. Jelikož je v dnešní době naprostá většina softwaru vyvíjena v rámci menších nebo větších týmů, lze považovat ovládnutí schopnosti pracovat v týmu za velmi podstatnou. Nedílnou součástí práce v týmu je také získání zkušeností s verzovacími systémy jako Git.

Jak je v jedné z předchozích kapitol zmíněno, jedním z mých významnějších nedostatků byla nezkušenost s prací na větších projektech, při kterých by byl brán zásadní ohled na návrh architektury. S tímto je spojena neznalost většiny principů softwarového vývoje a návrhových vzorů, jejichž rozsáhlé množství je nemožné probrat pouze v předmětech *Úvod do softwarového inženýrství* a *Vývoj informačních systémů*.

8 Závěr

V rámci praxe se nám podařilo dosáhnout většiny stanovených cílů. Byla vytvořena plně funkční hra pro virtuální realitu za využití herního enginu Unity. Výslednou hru je možné vydat na herní trh, k čemuž by mělo dojít po vypracování této práce. Byly splněny všechny zadané požadavky na hru a celkový ohlas ve firmě byl velmi pozitivní.

Osobně se mi taktéž podařilo úspěšně splnit všechny mi zadané úkoly. Realizoval jsem fungování hry pro virtuální realitu skrze knihovnu Virtual Reality Toolkit, implementoval jsem hlavní funkcionalitu spojenou s činností hráče ve hře a taktéž jsem vytvořil nespočet systémů pro řízení této hry. Taktéž jsem se pokusil o zlepšení kvality kódu ve hře za využití lepšího přístupu k programování a návrhových vzorů. A v neposlední řadě jsem se svými nápady a připomínkami podílel na výsledném designu hry.

Vytvořená hra by v budoucnu mohla být dále rozšířena přidáním dalšího herního obsahu. Může se jednat o další různorodé arény a typy nepřátel nebo zcela nové herní prvky. Častými body diskuze bylo například přidání bonusů z padlých nepřátel nebo boss soubojů, při kterých by hráč musel využít rozmanitých strategií pro jejich zničení. Jelikož je tvorba her velmi kreativní záležitost, možnosti jsou zde téměř neomezené. Je taky možné vytvořit verzi hry pro jiné platformy než SteamVR. Jelikož je engine Unity vysoce multiplatformní a knihovna Virtual Reality Toolkit podporuje vícero technologií virtuální reality, nemuselo by se jednat o příliš obtížný proces.

Absolvování této praxe považuji jako velmi pozitivní zkušenost. Za odborného vedení jsem se naučil základům využití enginu Unity a měl jsem možnost pracovat s virtuální realitou. Jako další významnou zkušenost považuji práci ve vícečlenném týmu s lidmi s jiným studijním zaměřením. Dále jsem pracoval na větším projektu, díky čemuž jsem pochopil nutnost potřeby využití principů správného programování a návrhových vzorů. Všechny tyto znalosti a zkušenosti považuji za nenahraditelné a jistě mi budou do budoucna k dobru. Díky této praxi mi jsou dostupné nové možnosti a příležitosti v oblasti herního vývoje, které bych se i nadále chtěl věnovat.

Literatura

[1] Craneballs [online]. [cit. 2019-03-09].
Dostupné z: https://www.craneballs.com/

[2] SteamVR [online]. [cit. 2019-03-09].
Dostupné z: https://steamcommunity.com/steamvr

[3] Unity [online]. [cit. 2019-03-09].
Dostupné z: https://unity3d.com/

[4] VRTK on Unity Asset Store [online]. [cit. 2019-03-16].
Dostupné z: https://assetstore.unity.com/packages/tools/integration/vrtk-virtual-reality-toolkit-vr-toolkit-64131

[5] Overkill on App Store [online]. [cit. 2019-03-09].
Dostupné z: https://itunes.apple.com/cz/app/overkill/id421659813?mt=8

[6] Bomb Hunters on Google Play [online]. [cit. 2019-04-28].
Dostupné z: https://play.google.com/store/apps/details?id=com.craneballs.bombhunter

[7] Bomb Hunters on Apple Store [online]. [cit. 2019-04-28].
Dostupné z: https://itunes.apple.com/app/bomb-hunters/id1164102837

[8] Splash Cars on Google Play [online]. [cit. 2019-04-28].
Dostupné z: https://play.google.com/store/apps/details?id=com.craneballs.artdrive

[9] Splash Cars on Apple Store [online]. [cit. 2019-04-28].
Dostupné z: https://itunes.apple.com/app/splash-cars/id1042273323

[10] Overkill 3 on Google Play [online]. [cit. 2019-03-09].
Dostupné z: https://play.google.com/store/apps/details?id=com.craneballs.overkill3

[11] Overkill 3 on App Store [online]. [cit. 2019-03-09].
Dostupné z: https://itunes.apple.com/app/overkill-3/id874866593

[12] Medieval Smackdown on Google Play [online]. [cit. 2019-03-16].
Dostupné z: https://play.google.com/store/apps/details?id=com.craneballs.smackdown

[13] Medieval Smackdown on Apple Store [online]. [cit. 2019-03-16].
Dostupné z: https://itunes.apple.com/cz/app/medieval-smackdown/id1255425988?mt=8

[14] Planet Nomads [online]. [cit. 2019-03-09].
Dostupné z: https://www.planet-nomads.com/

[15] Planet Nomads Kickstarter Campaign [online]. [cit. 2019-03-09].
Dostupné z: https://www.kickstarter.com/projects/2043603103/planet-nomads

[16] Tron: Legacy [online]. [cit. 2019-04-28].
Dostupné z: https://www.imdb.com/title/tt1104001/

[17] Sparc [online]. [cit. 2019-04-28].
Dostupné z: https://www.playsparc.com/

[18] Space Pirate Trainer [online]. [cit. 2019-04-28].
Dostupné z: https://www.spacepiratetrainer.com/

[19] Blue Effect [online]. [cit. 2019-04-28].
Dostupné z: https://www.blue-effect.com/

[20] Unity Supported Platforms [online]. [cit. 2019-03-09].
Dostupné z: https://unity3d.com/unity/features/multiplatform

[21] Unity Subscription Plans [online]. [cit. 2019-03-17].
Dostupné z: https://store.unity.com/?_ga=2.267760462.1956677625.1553535611-500210767.1530911447

[22] HAAS, John. A History of the Unity Game Engine [online]. [cit. 2019-03-16].
Dostupné z: https://web.wpi.edu/Pubs/E-project/Available/E-project-030614-143124/unrestricted/Haas_IQP_Final.pdf. An Interactive Qualifying Project. WORCESTER POLYTECHNIC INSTITUTE. Vedoucí práce Brian Moriarty, IMGD.

[23] UnityScript's long ride off into the sunset [online]. [cit. 2019-03-16].
Dostupné z: https://blogs.unity3d.com/2017/08/11/unityscripts-long-ride-off-into-the-sunset/

[24] VRTK on GitHub [online]. [cit. 2019-03-16].
Dostupné z: https://github.com/thestonefox/VRTK

[25] GitHub [online]. [cit. 2019-03-16].
Dostupné z: https://github.com/

[26] Sourcetree [online]. [cit. 2019-03-16].
Dostupné z: https://www.sourcetreeapp.com/

[27] Scrum [online]. [cit. 2019-03-09].
Dostupné z: https://www.scrum.org/resources/what-is-scrum

[28] Tech 101: What Exactly is Scrum? [online]. [cit. 2019-03-31].
Dostupné z: https://skillcrush.com/2017/06/28/what-is-scrum-project-management/

[29] The Beginner's Guide To Scrum And Agile Project Management [online]. [cit. 2019-03-31]. Dostupné z: https://blog.trello.com/beginners-guide-scrum-and-agile-project-management

[30] Pivotal Tracker [online]. [cit. 2019-04-28]. Dostupné z: https://www.pivotaltracker.com/

[31] The Benefits and Pitfalls of Pair Programming in the Workplace [online]. [cit. 2019-03-31]. Dostupné z: https://medium.freecodecamp.org/the-benefits-and-pitfalls-of-pair-programming-in-the-workplace-e68c3ed3c81f

[32] Pair Programming Guide [online]. [cit. 2019-03-31]. Dostupné z: https://medium.com/@weblab_tech/pair-programming-guide-a76ca43ff389

[33] Z virtuální reality vám asi bude špatně [online]. [cit. 2019-03-31]. Dostupné z: http://vtm.e15.cz/z-virtualni-reality-vam-asi-bude-spatne

[34] Virtuální realita se zadrhla? Problém her může být kinetóza. [online]. [cit. 2019-03-31]. Dostupné z: https://www.prochlapy.cz/clanky/virtualni-realita-se-zadrhla-problem-her-muze-byt-kinetoza-ma-vr-jeste-dalsi-vyuziti-a-slibnou-budoucnost/

[35] Movement in VR by Unity [online]. [cit. 2019-03-16]. Dostupné z: https://unity3d.com/learn/tutorials/topics/virtual-reality/movement-vr

[36] 7 Ways to Move Users Around in VR Without Making Them Sick [online] 1-2 [cit. 2019-03-16]. Dostupné z: https://www.roadtovr.com/7-ways-move-users-around-vr-without-making-sick/

[37] Do the Locomotion: The 19 Ways You Walk and Run in VR Games [online]. [cit. 2019-03-31]. Dostupné z: https://www.tomshardware.com/picturestory/807-virtual-reality-games-locomotion-methods.html#s15

[38] Connect: Developing VR Experiences with the Oculus Rift [online]. 29.10.2014 [cit. 2019-03-16]. Dostupné z: https://www.youtube.com/watch?v=addUnJpjjv4 &feature=youtu.be&t=40m5s. Kanál uživatele Oculus

[39] Physics Best Practices [online]. [cit. 2019-03-17]. Dostupné z: https://unity3d.com/learn/tutorials/topics/physics/physics-best-practices

[40] Unity: Now You're Thinking With Components [online]. [cit. 2019-04-10]. Dostupné z: https://gamedevelopment.tutsplus.com/articles/unity-now-youre-thinking-with-components–gamedev-12492

[41] Components in Unity [online]. [cit. 2019-04-10]. Dostupné z: http://gram.gs/gramlog/components-in-unity/

[42] Steam Refund Policy [online]. [cit. 2019-03-17].
Dostupné z: https://store.steampowered.com/steam_refunds/